1. Conditional Expressions and Writing Functions

Conditional Expressions

- `EXPRESSION` runs only if `CONDITION` is `TRUE`. (Here `UPPER.CASE` is a placeholder for R code.)

```
if (CONDITION) {
  EXPRESSION
}
```

- One of `TRUE.EXPRESSION` and `FALSE.EXPRESSION` runs:

```
if (CONDITION) {
  TRUE.EXPRESSION
} else {
  FALSE.EXPRESSION
}
```

Note that "`} else`" must be on a single line (for console, source()).

```
# example
x = 3
if ((x %% 2) == 0) {
  parity = "even"
} else {
  parity = "odd"
}
parity
```

- The first true `CONDITION`'s `EXPRESSION` runs; or, if none is true, `DEFAULT_EXPRESSION` runs:

```
if (CONDITION_1) {
  EXPRESSION_1
} else if (CONDITION_2) { # optional "else if" clauses
  EXPRESSION_2            # ...
} ... {
  ...
} else {                  # optional "else" clause
  DEFAULT_EXPRESSION
}
```

```
temperature = 60                # example
if (temperature < 32) {
  state = "frozen"
} else if (temperature > 212) {
  state = "boiling"
} else {
  state = "liquid"
}
cat(sep="", "water is ", state, "\n")
```

- None of the three constructs above works if its `CONDITION` has length greater than one. However, `x = ifelse(test, yes, no)` sets `x` to be a vector with the same length as `test` filled with elements from `yes` or `no` depending on the logical values in `test`. e.g.

```
parity = ifelse((x %% 2) == 0, "even", "odd")
```

Writing functions

```
FUNCTION.NAME = function(PARAMETER.LIST) {
  BODY
}
```

(See `day1.R` for examples.)

A function call proceeds as follows:

- Execution jumps to first line of function upon seeing the call, `FUNCTION.NAME(ARGUMENT.LIST)`
- Function's `PARAMETER.LIST` is *copied* from caller's `ARGUMENT.LIST` by name or position, and from defaults specified as `PARAMETER.NAME=DEFAULT` in `PARAMETER.LIST`
- Assignment to function parameters and local variables doesn't affect caller's variables
- Code in function is executed until `return(EXPRESSION)`, or until function's closing `}`
- Execution returns to caller; if caller assigned a variable to function, it gets `EXPRESSION` from function's `return()` or last expression

  Note: `return()` and `cat()` are not the same thing. `return()` returns a value to which the caller can assign a variable, which affects the state of the program. `cat()` (like `print()`) writes text on the console, which can be helpful to a human reader, but it doesn't affect the state of the program. Typically a function should use `return()`, not `cat()`, to provide its output. e.g. `x = sqrt(16)` vs. `sqrt(16)`.

e.g. Here's a strange example to illustrate the points above:

```
square.a = function(a=1, b=2) {
  cat(sep="", "  square.a(a=", a, ", b=", b, ")\n")
  b = 100
  c = a*a
  return(c)
}

square.a(a=3, b=4) # two identical calls
square.a(b=4,   3)
a = 5; b = 6; c = 7
square.a(b)
cat(sep="", "a=", a, ", b=", b, ", c=", c, "\n")
```

(Hint: study the bullet points on this page again after completing hw1.R.)